Objektorientierte Datenbanken

Vorlesung 4 vom 29.04.2004

Dr. Sebastian Iwanowski
FH Wedel

Korrektur zu Folie 18 von OODB3:

Class Extent

public Extent (Database db, String className);

➔ *public Object next ();* Nicht void !

public boolean hasNext ();

viele weitere Methoden . . .

Inhalt heute:

Die ODMG-Anfragesprache OQL:

Wozu brauchen wir eine Anfragesprache ?

Wesentliche Elemente von OQL

FastObjects-OQL

Vertiefende Beispiele

Wozu brauchen wir eine Anfragesprache ?

Datenbankanfragemöglichkeiten ohne Anfragesprache

a) Extraktion eines einzelnen Objects (spezifiziert durch Name)

b) Extraktion der Extension einer Klasse

Ungeeignet für das Problem:

Finde Objekte mit bestimmten Eigenschaften !

a) Hoher Programmieraufwand

b) Hoher Hauptspeicherbedarf

Motivation für Objektanfragesprache

- **gezielte Extraktion von Objekten mit bestimmten Eigenschaften**

  mehr Komfort, mehr Effizienz

- **Abfragemöglichkeiten ohne detaillierte Java-Kenntnisse**

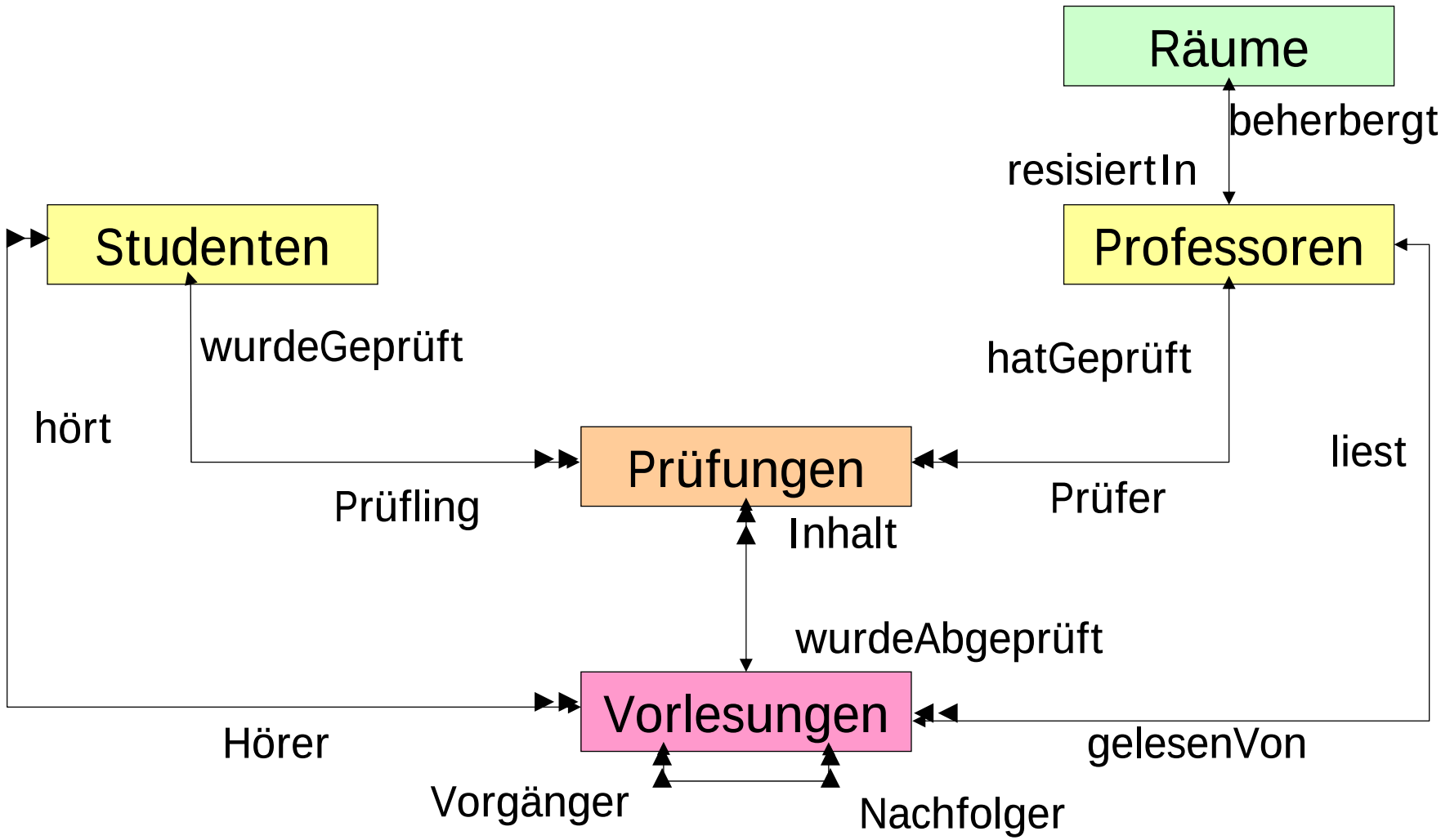
  politischer und sozialer Grund

Historie von OQL

- **implementiert vom Datenbankhersteller O_2 ca. 1990**
- **aufgenommen in den ODMG-Standard als konkurrenzloser Kandidat**

Wesentliche Elemente von OQL

Modell für die kommenden Beispiele



Modell für die kommenden Beispiele

```
class Professoren {
        attribute long PersNr;
        attribute string Name;
        attribute string Rang;
        relationship Räume residiertIn inverse Räume::beherbergt;
        relationship set(Vorlesungen) liest inverse Vorlesungen::gelesenVon;
        relationship set(Prüfungen) hatGeprüft inverse Prüfungen::Prüfer;
};

class Vorlesungen {
        attribute long VorlNr;
        attribute string Titel;
        attribute short SWS;
        relationship Professoren gelesenVon inverse Professoren::liest;
        relationship set(Studenten) Hörer inverse Studenten::hört;
        relationship set(Vorlesungen) Nachfolger inverse Vorlesungen::Vorgänger;
        relationship set(Vorlesungen) Vorgänger inverse Vorlesungen::Nachfolger;
        relationship set(Prüfungen) wurdeAbgeprüft inverse Prüfungen::Inhalt;
};
```

Modell für die kommenden Beispiele

```
class Studenten {
        attribute string Name;
        attribute boolean female;
        attribute int Fachsemester;
        relationship set(Vorlesungen) hört inverse Vorlesungen::Hörer;
        relationship set(Prüfungen) wurdeGeprüft inverse Prüfungen::Prüfling;
};


class Prüfungen {
    attribute struct Datum
        { short Tag; short Monat; short Jahr} Prüfdatum;
    attribute float Note;
    relationship Professoren Prüfer inverse Professoren::hatGeprüft;
    relationship Studenten Prüfling inverse Studenten::wurdeGeprüft;
    relationship Vorlesungen Inhalt inverse Vorlesungen::wurdeAbgeprüft;
};

Class Räume {
     attribute string RaumId;
     relationship Professoren beherbergt inverse Professoren::residiertIn;
```

Direkte Anfragen

☹ **wird in FastObjects nicht berücksichtigt** ☹

Anfragen ohne select-from-where-Struktur:

- Finde den Raum des Professors mit Objektbezeichner Chef:

```
Chef.residiertIn.raumId
```

- Finde Name und Raum des Professors mit Objektbezeichner Chef:

```
struct (name: Chef.Name, raum: Chef.residiertIn.raumId)
```

- Anfragen spiegeln die ursprünglichen Objektdefinitionen von O_2 wieder
- Objektausdrücke werden als Funktion interpretiert und bekommen als Werte die zugehörigen Datenbankeinträge

SQL-ähnliche Anfragen

Einfache Beispiele:

Fragen nach allen Daten mit gemeinsamen Attributen:

- Finde die Namen der C4-Professoren

```
select p.Name
from p in AlleProfessoren
where p.Rang = "C4"
```

```
select p.Name
from AlleProfessoren as v
where v.Rang = "C4"
```

Fragen nach zusammengesetzten Objekten: ☹ wird in FastObjects nicht berücksichtigt ☹

- Finde Name und Raum aller C4-Professoren

```
select struct(n: p.Name, r: p.residiertIn)
from p in AlleProfessoren
where p.Rang = "C4"
```

Geschachtelte Anfragen

☹ wird in FastObjects nicht berücksichtigt ☹

- Finde alle Professoren, die lange Vorlesungen halten:

```
select struct(n: p.Name, a: sum(select v.SWS from v in p.liest))
from p in AlleProfessoren
where max(select v.SWS from v in p.liest) >= 4
```

Partitionierung

☹ wird in FastObjects nicht berücksichtigt ☹

- Teile Vorlesungen in kurze, mittlere und lange ein:

```
select *
from v in AlleVorlesungen
groupBy kurz: v.SWS <= 2, mittel: v.SWS = 3, lang v.SWS >= 4
```

Navigieren in Objektpfaden

- Finde die Namen aller Studenten, die von Sokrates geprüft wurden:

```
select s.Name
from s in AlleStudenten
where s.wurdeGeprüft.Prüfer.Name = „Sokrates“
```

- Finde die Namen aller Studenten, die bei Sokrates Vorlesung haben:

```
select s.Name
from s in AlleStudenten, v in s.hört
where v.gelesenVon.Name = „Sokrates“
```

- Finde die Namen aller weiblichen Studenten, die bei Sokrates Vorlesung haben:

```
select s.Name
from s in AlleStudenten, v in s.hört
where (v.gelesenVon.Name = „Sokrates“) and s.female
```

Navigieren in Objektpfaden

☹ das geht nicht in FastObjects: ☹

- Finde die Titel der Vorlesungen, in denen weibliche Studenten sitzen und die von Sokrates gehalten werden:

```
select v.Titel
from s in AlleStudenten, v in s.hört
where (v.gelesenVon.Name = „Sokrates“) and s.female
```

☺ so geht es: ☺

- Finde die Titel der Vorlesungen, in denen weibliche Studenten sitzen und die von Sokrates gehalten werden:

```
select v.Titel
from v in AlleVorlesungen, s in v.Hörer
where (v.gelesenVon.Name = „Sokrates“) and s.female
```

„Selection of collection members not supported yet“

Rückgabe von vollständigen Objekten

Ziel: Objekte sollen in Programmiersprache weiterbearbeitet werden können

- Finde alle Studenten, die von Sokrates geprüft wurden:

```
select s
from s in AlleStudenten
where s.wurdeGeprüft.Prüfer.Name = „Sokrates“
```

Integration von Objekt-Methoden

☹ wird in FastObjects nicht berücksichtigt ☹

- Finde alle Professoren, deren Gehalt größer als 50000 € ist :

```
select p
from p in AlleProfessoren
where p.Gehalt() > 50000
```

Umgang mit Polymorphismus

☹ wird in FastObjects nicht berücksichtigt ☹

- Finde alle Angestellten, deren Gehalt größer als 50000 € ist :

```
select a
from a in AlleAngestellten
where a.Gehalt() > 50000
```

Problem:

Angestellte, Professoren und Assistenten implementieren unterschiedliche Methoden Gehalt()

➔ späte Bindung der Methoden erforderlich

FastObjects-OQL

Einschränkungen von FastObjects-OQL

- **keine direkten Anfragen**

  nur select-from-where-Anfragen

- **keine Integration von Methodenaufrufen**

  Polymorphismusproblem existiert im Prinzip auch bei Attributen !

- **weitere Einschränkungen von OQL-Funktionalitäten wie zuvor gekennzeichnet:** ☹ wird in FastObjects nicht berücksichtigt ☹

- **weitere Einschränkungen von OQL-Funktionalitäten, die hier nicht besprochen wurden**

Wie werden Anfragen konkret gestellt ?

Class **OQLQuery**:

Konstruktoren:

- **`OQLQuery()`**
- **`OQLQuery(String query)`** *der „normale“ Konstruktor*
- **`OQLQuery (Transaction tn)`**
- **`OQLQuery (Database db)`**

Methoden:

- **`void create(String query)`** *nötig bei Nichtverwendung des normalen Konstruktors*
- **`Object execute()`** *immer nötig: stellt die Anfrage*

Grundsatz:

Jede OQLQuery muss einer Transaktion und einer Datenbank zugeordnet sein

Existenzquantoren

Beispiel für Existenzquantor:

- Finde die Titel der Vorlesungen, in denen weibliche Studenten sitzen und die von Sokrates gehalten werden:

```
select v.Titel
from v in AlleVorlesungen
where (v.gelesenVon.Name = „Sokrates“) and (exists s in v.Hörer: s.female)
```

Zur Erinnerung: So ging es ohne Existenzquantor:

```
select v.Titel
from v in AlleVorlesungen, s in v.Hörer
where (v.gelesenVon.Name = „Sokrates“) and s.female
```

Allquantoren

Beispiel für Allquantor:

- Finde die Titel der Vorlesungen, in denen nur weibliche Studenten sitzen und die von Sokrates gehalten werden:

select v.Titel

from v **in** AlleVorlesungen

where (v.gelesenVon.Name = „Sokrates“) and (**for all** s in v.Hörer: s.female)

Weitere Funktionalitäten

Sortierfunktion mit **`order by`**:

- Finde alle Studenten, die von Sokrates geprüft wurden:

```
select s
from s in AlleStudenten
where s.wurdeGeprüft.Prüfer.Name = „Sokrates“
order by s.Semesterzahl DESC, s.Name ASC
```

Zählfunktion mit **`count`**:

- Ordne die Vorlesungen von Sokrates nach der Zahl der Hörer:

```
select v.Titel
from v in AlleVorlesungen
where (v.GelesenVon.Name = „Sokrates“)
order by count (select s from s in v.Hörer) DESC
```

Zusammenfassung:
Eine Fragestellung mit OQL in FastObjects

```
db = new Database ();
db.open("FastObjects://LOCAL/ToolsBase", Database.OPEN_READ_WRITE );
Transaction txn = new Transaction( db );
txn.begin();
// Frage formulieren:
String queryString = "SELECT c FROM ToolkitExtent AS c " +
                                        "WHERE c.year_ = 1997";
// Frageobjekt erzeugen:
OQLQuery query = new OQLQuery( queryString );
// Frage stellen:
Object result = query.execute();
// Ergebnis auswerten . . .
txn.abort();
db.close()
```

Vertiefende Beispiele (nicht alles FastObjects-konform!)

Beispiel (1)

```
select a
    from a in AuthorExtent
    where exists p in a.publication:
                count(p.keywords) > 10

select p.publkey
    from p in flatten(select a.publications
                      from AuthorExtent a
                      where a.name like "G*")

element(select a from a in AuthorExtent
            where a.name = "Dittrich")
```

Beispiel (2)

```
max(
        select p.accesses
        from p in PublicationExtent)

select struct (author1: a, author2: b, publication: p)
     from PublicationsExtent p,
             a in p.getAuthors(),
             b in p.getAuthors()
     where a.name < b.name
```

Beim nächsten Mal:
Einführung in JDO